

# Modular Verification of Autonomous Space Robotics

Marie Farrell, Rafael C. Cardoso, Louise A. Dennis, Clare Dixon, Michael Fisher,  
Georgios Kourtis, Alexei Lisitsa, Matt Luckcuck and Matt Webster

Department of Computer Science  
University of Liverpool  
marie.farrell@liverpool.ac.uk

**Abstract**—Ensuring that autonomous space robot control software behaves as it should is crucial, particularly as software failure in space often equates to mission failure and could potentially endanger nearby astronauts and costly equipment. To minimise mission failure caused by software errors, we can utilise a variety of tools and techniques to verify that the software behaves as intended. In particular, distinct nodes in a robotic system often require different verification techniques to ensure that they behave as expected. This paper introduces a method for integrating the various verification techniques that are applied to robotic software, via a First-Order Logic (FOL) specification that captures each node’s assumptions and guarantees. These FOL specifications are then used to guide the verification of the individual nodes, be it by testing or the use of a formal method. We also outline a way of measuring our confidence in the verification of the entire system in terms of the verification techniques used.

**Keywords**—formal methods; heterogeneous verification; autonomous space robotics

## I. INTRODUCTION

Robotic systems combine many hardware and software components, usually represented as node-based architectures. Each node in a robotic system may require different verification techniques, ranging from software testing to formal methods. In fact, *integrating* (formal and non-formal) verification techniques is crucial for the robotics domain [2]. Verification should be carried out using the most suitable technique or formalism for each node. However, linking heterogeneous verification results of individual nodes is difficult and the current state-of-the-art for robotic software development does not provide an easy way of achieving this.

In Fig. 1, we consider a simple space robotic system: a planetary rover undertaking a remote inspection task. Here, we have nodes representing the **Vision** system, a **Planner** that returns a set of potential plans between the current location and the next point to inspect, an autonomous **Plan Reasoning Agent** that selects a plan, and a **Hardware Interface** that sends commands to the rover’s actuators.

As illustrated by Fig. 1, we could use logical specifications (e.g. temporal logic), model-based specifications

(e.g. Event-B or Z), or algebraic specifications (e.g. CSP or CASL) amongst others to specify the nodes in a robotic system. Each of these formalisms offers its own range of benefits, and each tends to suit the verification of particular types of behaviour. However, in some cases we may only have access to the black-box or white-box implementation of a node and so, we must use (simulation-based) testing techniques for verification.

Our approach facilitates the use of heterogeneous verification techniques for the nodes in a robotic system. We achieve this by specifying Assume-Guarantee [3] properties in FOL, as high-level node specifications, and we employ temporal logic for reasoning about the combination of these FOL specifications. Thus, we attach the assumptions ( $\mathcal{A}(\bar{i})$ ) and guarantees ( $\mathcal{G}(\bar{o})$ ) to individual nodes (shown in Fig. 1). This abstract specification can be seen as a logical prototype for individual nodes and thus the entire robotic system.

## II. FOL ASSUME-GUARANTEE SPECIFICATIONS

For each node,  $N$ , we specify  $\mathcal{A}_N(\bar{i}_N)$  and  $\mathcal{G}_N(\bar{o}_N)$  where  $\bar{i}_N$  is a variable representing the input to the node,  $\bar{o}_N$  is a variable representing the output from the node, and  $\mathcal{A}_N(\bar{i}_N)$  and  $\mathcal{G}_N(\bar{o}_N)$  are FOL formulae describing the assumptions and guarantees, respectively, of this node.

Each individual node,  $N$ , obeys the following implication

$$\forall \bar{i}_N, \bar{o}_N \cdot \mathcal{A}_N(\bar{i}_N) \Rightarrow \Diamond \mathcal{G}_N(\bar{o}_N)$$

where ‘ $\Diamond$ ’ is LTL’s [4] “eventually” operator. So, this implication means that if the assumptions,  $\mathcal{A}_N(\bar{i}_N)$ , hold then *eventually* the guarantee,  $\mathcal{G}_N(\bar{o}_N)$ , will hold. Note that our use of temporal operators here is motivated by the temporal nature of robotic systems and will be of use in later extensions of this work.

Consider the autonomous **Plan Reasoning Agent** in Fig. 1, we can specify the following simple assumption,  $\mathcal{A}_3(\bar{i}_3)$ :

$$\mathcal{A}_3(\bar{i}_3) = \forall p \cdot p \in PlanSet \Rightarrow goal \in p$$

which ensures that every plan that is returned by the **Planner** contains the *goal* location. Then, we might specify the guarantee that the agent chooses the shortest *plan* as follows:

$$\begin{aligned} \mathcal{G}_3(\bar{o}_3) &= plan \in PlanSet \\ &\wedge \forall p \cdot p \in PlanSet \wedge p \neq plan \\ &\Rightarrow length(plan) \leq length(p) \end{aligned}$$

This work is supported by grant EP/R026092 (FAIR-SPACE Hub) through UKRI under the Industry Strategic Challenge Fund (ISCF) for Robotics and AI Hubs in Extreme and Hazardous Environments.

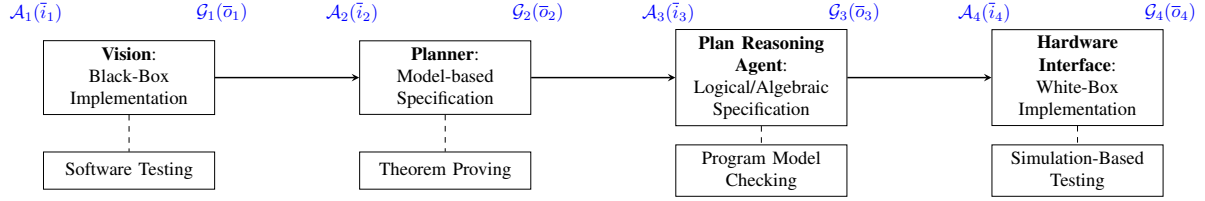


Figure 1. We specify the Assume-Guarantee properties for each node (denoted by  $\mathcal{A}_i(\bar{i}_i)$  and  $\mathcal{G}_i(\bar{o}_i)$  respectively). These are then used to guide the verification approach applied to each node, denoted by dashed lines, such as software testing for a black-box implementation of the **Vision** node. The solid arrows represent data flow between nodes and that the assumptions of the next node should follow from the guarantee of the previous.

	Testing	Simulation-Based Testing	Formal Methods
<b>Vision</b>	✓	✗	✗
<b>Planner</b>	✓	✓	✓
<b>Plan Reasoning Agent</b>	✓	✓	✓
<b>Hardware Interface</b>	✓	✓	✗

Table I  
VERIFICATION TECHNIQUES APPLIED TO EACH NODE.

Once the FOL assumption and guarantee are specified, then we use these high-level specifications as properties to be verified of the individual nodes. For the autonomous **Plan Reasoning Agent**, we can use a number of techniques for verifying that it meets its associated FOL specification. For example, we can specify the node using the GWENDOLEN agent programming language and then use the AJPF model-checker to verify that it behaves as specified [1].

Nodes in a modular robotic architecture are linked together and transmit data between them so long as their types/requirements match. Similarly, we can compose the assume-guarantee specifications of individual nodes in a number of ways and we are working towards a calculus of inference rules that capture this behaviour. To this end, we are developing rules for sequentially composing, joining, branching and looping between nodes.

### III. MEASURING CONFIDENCE IN VERIFICATION

A key question is how using these different verification techniques affects our confidence in the verification of the whole system. One might think that a formal proof of correctness corresponds to a higher level of confidence than simple testing methods (especially over unbounded environments). However, formal verification is usually only feasible on an abstraction of the system whereas testing can be carried out on the implemented code. Therefore, it is our view that we achieve higher levels of confidence in verification when multiple verification methods have been employed for each node in the system [5].

We have broadly partitioned current verification techniques into three categories: testing, simulation-based testing and formal methods. We have determined which of these techniques might be employed for each node in our simple example as shown in Table I. We then provide a score for our

level of confidence in the verification of the whole system as 9/12, resulting in a confidence measure of 75%. Examining how this metric can be calculated for more complex systems with loops is a future direction for this work.

### IV. CONCLUSIONS

When verifying complex robotic systems, it is clear that no single verification technique is suitable for every node in the system [2] and so a logical framework that allows us to integrate the results from distinct verification techniques is needed. We have outlined an initial approach to specifying assumptions and guarantees using FOL for individual nodes in robotic systems and we have used a simple, illustrative example of a planetary rover to convey our approach. Once the FOL specifications have been constructed, they are then used to guide the more detailed verification of each node. Furthermore, we introduce the notion of confidence in verification techniques and provide a broad categorisation.

Our current work involves developing a calculus for reasoning about and combining the Assume-Guarantee specifications of individual nodes. In the future, we plan to provide tool support for this and to evaluate it using a set of more complex robotic space missions. We also intend to further investigate the suitability of the confidence levels that we have proposed in this paper.

### REFERENCES

- [1] L. A. Dennis, M. Fisher, M. P. Webster, and R. H. Bordini. Model checking agent programming languages. *Automated Software Engineering*, 19(1):5–63, 2012.
- [2] M. Farrell, M. Luckcuck, and M. Fisher. Robotics and Integrated Formal Methods: Necessity meets Opportunity. In *Integrated Formal Methods*, volume 11023 of *LNCS*, pages 161–171. Springer, 2018.
- [3] C. B. Jones. Tentative Steps Toward a Development Method for Interfering Programs. *ACM Transactions on Programming Languages and Systems*, 5(4):596–619, 1983.
- [4] A. Pnueli. The Temporal Logic of Programs. In *18th Symposium on the Foundations of Computer Science*, pages 46–57. IEEE, 1977.
- [5] M. Webster, D. Western, D. Araiza-Illan, C. Dixon, K. Eder, M. Fisher, and A. G. Pipe. A corroborative approach to verification and validation of human–robot teams. *arXiv preprint arXiv:1608.07403*, 2016.